FILE COPY

(2)

AD-A226 738

Capturing Strong Reduction in Director String Calculus

DTIC
ELECTE
AUG 3 0 1990
S D
D

by

Vugranam Sreedhar and Kazem Taghva

Department of Computer Science [1]
University of Nevada, Las Vegas

July 5, 1990

CSR-90-38

Abstract

A modified version of Director String Calculus (MDSC) is introduced which preserves the applicative structure of the original lambda terms and captures the strong reduction as opposed to weak reduction of the original Director String Calculus (DSC). Futhermore, MDSC provides an environment which supports the nonatomic nature of the substitution operation and hence can lend itself to parallel and optimal reduction. (KR) ⌐

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

[1]This research was supported in part by U. S. Army Research Office under grant DAAL03-87-G-0004

90 08 27 072

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 1990	3. REPORT TYPE AND DATES COVERED Technical
4. TITLE AND SUBTITLE Capturing Strong Reduction in Director String Calculus		5. FUNDING NUMBERS DAAL03-87-G-0004
6. AUTHOR(S) Vugranam Sreedhar and Kacem Taghva		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Univ of Nevada Las Vegas, Nevada 89154-4019		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U. S. Army Research Office P. O. Box 12211 Research Triangle Park, NC 27709-2211		10. SPONSORING/MONITORING AGENCY REPORT NUMBER ARO 24960.52-MA-REP

11. SUPPLEMENTARY NOTES
The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A modified version of Director String Calculus (MDSC) is introduced which preserves the applicative structure of the original lambda terms and captures the strong reduction as opposed to weak reduction of the original Director String Calculus (DSC). Futhermore, MDSC provides an environment which supports the nonatomic nature of the substitution operation and hence can lend itself to parallel and optimal reduction.

14. SUBJECT TERMS	15. NUMBER OF PAGES 28
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

1 Introduction

Combinatory logic offers a practical approach to the implementations of functional languages [6, 12]. Generally, the implementations involve removal of variables from the text by introducing abstractions such as λ, and then translating the new text into combinatory logic [3, 15]. However, as O'Donnell and Strandh [11] point out, the translation from λ-calculus to combinatory logic disables some of the redexes present in the original term; and consequently avoids the possibility of doing parallel reductions. Furthermore, the size of the term in the combinatory logic tends to be larger than the corresponding λ-term. It is also apparent from the literature that one of the main problems in this approach is the operation of substitution in β-reduction [11, 13].

Kennaway and Sleep [8] introduced director strings as combinators to formally study the approach taken by Turner. Director strings also provide an intuitive interpretation of the reduction rules of the combinators to explain what combinators achieve. Director string implementation of λ-calculus as shown in [8] involves *weak* reduction according to Hindley et al [5]. In this paper, we introduce a modified version of director string calculus and obtain the *strong* reduction. In addition, our approach intuitively supports the nonatomic nature of substitutiion as explained in [11, 13] and lends itself to parallel and optimal reduction [4, 7, 9, 10, 14].

We assume that the readers are familiar with [8] throughout the rest of this paper.

2 Preliminaries

In this section, we will introduce some notations and preliminaries which will be used throughout the rest of the paper. As usual [1], let VAR be the set of variables, then the set of λ-terms Λ is defined inductively to be:

1. x is a term, where $x \in VAR$.
2. (EF) is a term, where $E, F \in \Lambda$.
3. $\lambda x.E$ is a term, where $x \in VAR$, and $E \in \Lambda$.

An occurence of a variable x in a λ-term E is *bound* if it is inside a subterm of the form $\lambda x.F$, otherwise it is free. We will use subscripts to encode the freeness or otherwise of the variable x in a term. For example, E_x denotes the λ-term E with free variable x. Kennaway and Sleep [8] introduced director symbols $\wedge, /, \backslash, -$ to abstract a variable from an application using the following six rules:

1. $\lambda x.(E_x F_x) \rightarrow \wedge((\lambda x.E_x)(\lambda x.F_x))$
2. $\lambda x.(E_x F) \rightarrow /((\lambda x.E_x)F)$
3. $\lambda x.(EF_x) \rightarrow \backslash(E(\lambda x.F_x))$
4. $\lambda x.(EF) \rightarrow -(EF)$
5. $\lambda x.x \rightarrow \mathbf{I}$

Accesion For		
NTIS CRA&I		☑
DTIC TAB		☐
Unannounced		☐
Justification		
By		
Distribution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

QUALITY INSPECTED 4

6. $\lambda x.y \rightarrow (\mathrm{K}y)$, where K and I are the standard combinators

Here, the objective is to move abstraction through an application, leaving a directing symbol behind. For example, the second rule says, the abstraction of x from a combination in which x occurs free in E_x but not in F can be encoded as a "send to the left" director symbol, denoted by /. The graph notation [12] represents a natural interpretation for these rules. For example, the term $\lambda x \lambda y.xy$ is transformed using the above rules in figure 1.

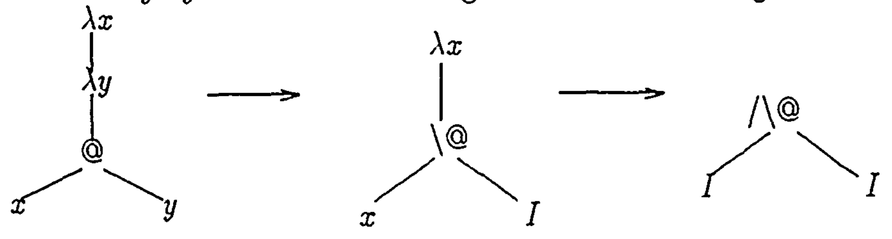


Figure 1

and the λ-term $\lambda x.((\lambda y.xy)x)$ is transformed in figure 2.

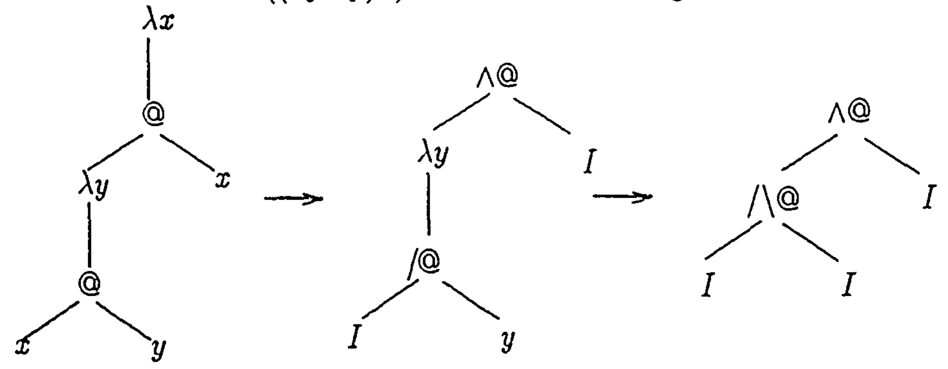


Figure 2

Director string calculus is defined as a term rewriting system with symbols from $VAR \cup \{\wedge, /, \backslash, -, \#, \Delta\}$ together with pair and triple constructors. Informally, the three unary operators $\#, !,$ and Δ are interpreted as "discard", "insert", and "hole" respectively, and the four binary operators $\wedge, /, \backslash$ and $-$ are interpreted as "both ways", "send to left", "send to right", and "neither way" respectively. Following [8], the λ-terms $\lambda x.\lambda y.xy$ and λ-term $\lambda x.((\lambda y.xy)x)$ are written as

$$(/\backslash, (!, \Delta), (!, \Delta))$$

and

$$(\wedge, (/\backslash, (!, \Delta), (!, \Delta)), (!, \Delta))$$

respectively. Observe that the last two expressions represent the trees in figure 1 and figure 2 with I's replaced by $(!, \Delta)$. Now, let $@_1$, $@_2$, d, and D denote empty unary, empty binary, arbitrary director symbol, and arbitrary director string respectively. Then the rules of [8] defines how the terms are reduced in director string calculus.

1. $(@_2, (\wedge D, E_1, E_2), E_3) \rightarrow (D, (@_2, E_1, E_3), (@_2, E_2, E_3))$

2. $(@_2, (/D, E_1, E_2), E_3) \rightarrow (D, (@_2, E_1, E_3), E_2)$

3. $(@_2, (\backslash D, E_1, E_2), E_3) \rightarrow (D, E_1, (@_2, E_2, E_3))$

4. $(@_2, (-D, E_1, E_2), E_3) \rightarrow (D, E_1, E_2)$

5. $(@_2, (!D, E_1), E_2) \rightarrow (D, (@_2, E_1, E_2))$

6. $(@_2, (\#D, E_1), E_2) \rightarrow (D, E_1)$

7. $(@_2, \Delta, E_1) \rightarrow E_1$

Now, the expression $(\lambda x.\lambda y.xy)EF$ can be reduced to (EF) with the above conversion rules. String director calculus is not capable of strong reduction as our second example $\lambda x.((\lambda y.xy)x))$ can not be reduced further, although it can be reduced in λ-calculus to $\lambda x.xx$.

In the next section, we will give a modified string director calculus which is capable of strong reduction.

3 Modified Director String Calculus

Modified Director String Calculus (MDSC) has the same set of unary operators $\{!, \#, \Delta\}$ and four binary operators $\{/\backslash, -\backslash, /-, --\}$ together with pair and triple constructors. The binary director symbols $\{/\backslash, /-, -\backslash, --\}$ have the natural interpretations *send both to left and right, send to left but not right, send to right but not to left, send neither to left nor to right.* Although we are at liberty to choose one symbol for each binary operator, we prefer to use two symbols and intuitively think of them as the right side and left side of the binary operator. In addition, these two symbol operators make it easier to explain our evaluation strategies. We will basically follow the notation of [8] and let $D, D_1, D_2, d, @_1$ and $@_2$ denote arbitrary director string, director string over unary operator, director string over binary operator, arbitrary director symbol, empty director string over unary operators, and empty director string over binary operators respectively. We define Modified Director String Terms(MDST) as follows:

- $(D_1, a) \in MDST$, where a is either a variable or Δ
- $(D_1, E) \in MDST$, where $E \in MDST$
- $(D_2, E_1, E_2) \in MDST$, where $E_1, E_2 \in MDST$

In order to go back and forth between λ-calculus and MDSC, we utilize a mixed λ-calculus and Director String terms ($MDS\lambda T$). We basically map each term in λ-calculus to a term in $MDS\lambda T$ by replacing each variable x by $(@_1, x)$ and every application (EF) by $(@_2, E, F)$. This as in [8] embeds the λ-calculus into a system $MDS\lambda$ generated by the syntactic rules:

- $E \in MDST \Rightarrow E \in MDS\lambda T$
- $E \in MDST \Rightarrow \lambda x.E \in MDS\lambda T$, where x is a variable.

Thus $\lambda f \lambda x.f(fx)$ converts to the mixed term:

$$\lambda f \lambda x.(@_2, (@_1, f), (@_2, (@_1, f), (@_1, x)))$$

Then we define a translation from λ-terms in $MDS\lambda T$ to $MDST$, which removes all λ's and bound variables.

- $\lambda x.(D_1, x) \rightarrow (!D_1, \Delta)$
- $\lambda x.(D_1, \Delta) \rightarrow (\#D_1, \Delta)$
- $\lambda x.(D_1, y) \rightarrow (\#D_1, y)$, where $x \neq y$
- $\lambda x.(D_2, E_x, F_x) \rightarrow (/\backslash D_2, (\lambda x.E_x), (\lambda x.F_x))$

- $\lambda x.(D_2, E_x, F) \rightarrow (/ - D_2, (\lambda x.E_x), (\lambda x.F))$

- $\lambda x.(D_2, E_x, F_x) \rightarrow (-\backslash D_2, (\lambda x.E), (\lambda x.F_x))$

- $\lambda x.(D_2, E_x, F_x) \rightarrow (-- D_2, (\lambda x.E), (\lambda x.F))$

Example 3.1 *The term $\lambda f \lambda x.f(fx)$ will be written as follows:*

$$\begin{aligned}
\lambda f \lambda x.(@_2, (@_1, f), (@_2, (@_1, f), (@_1, x))) &\rightarrow \\
\lambda f.(-\backslash, \lambda x.(@_1, f), \lambda x.(@_2, (@_1, f), (@_1, x))) &\rightarrow \\
\lambda f.(-\backslash, (\#, f), (-\backslash, \lambda x.(@_1, f), \lambda x.(@_1, x))) &\rightarrow \\
\lambda f.(-\backslash, (\#, f), (-\backslash, (\#, f), (!, \Delta))) &\rightarrow \\
(/\backslash - \backslash, \lambda f.(\#, f), \lambda f.(-\backslash, (\#, f), (!, \Delta))) &\rightarrow \\
(/\backslash - \backslash, (!\#, \Delta), (/ - -\backslash, \lambda f.(\#, f), \lambda f.(!, \Delta))) &\rightarrow \\
(/\backslash - \backslash, (!\#, \Delta), (/ - -\backslash, (!\#, f), (\#!, \Delta)))
\end{aligned}$$

One may represent the last expression as the tree shown in figure 3.

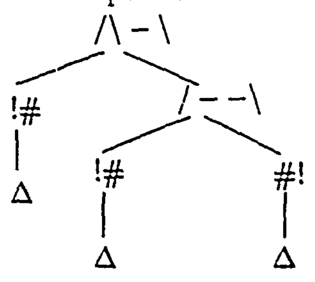


Figure 3

Intuitively, the tree explains how the evaluation must proceed, i.e. starting at the root the first argument to this expression will go both ways, the second argument to right but not to left, at the next tree level, for example, the left subtree says insert the first argument and discard the second argument. We will continue to use the tree structure to explain the conversion rules. We also want to point out that the translation does preserve the applicative structure of the original λ-term.

Proposition 3.1 *The abstraction rules are confluent and terminating. Every λ-term has a unique normal form, and this normal form is a MDS term.*

Proof: Similar to the proof of Proposition 2.2.1 in [8].

□

4 Conversion Rules

In this section, we will introduce our conversion rules and give detailed examples to illustrate the difference between MDS and the original DS. Before we formally write our conversion rules, we need three basic operations, namely *insert*, *shift*, and *remove*.

Let $E \in MDS$ and i, j be natural numbers, then:

- $shift(E, i, j)$ is defined to be a term $F \in MDS$ obtained from E by shifting $(i+1)$th director j places to the right for each director string in E.

- $insert(E, i, j)$ is also defined to be a term $G \in MDS$ obtained from E by inserting j numbers of $--$ between $(i+1)$ and $(i+2)$ positions of each binary director string E, and inserting j numbers of # between $(i+1)$ and $(i+2)$ positions of each unary director string in E.

- *remove*(E,i) is defined to be a term $H \in MDS$ obtained from E by removing $(i+1)$st director of every director in E.

Example 4.1 *Let E denote* $(-\backslash--/-,(\#\#!,\Delta),(!\#\#,\Delta))$, *then*

$$\begin{aligned} shift(E,1,1) &= (-\backslash/---,(\#!\#,\Delta),(!\#\#,\Delta)) \\ insert(E,1,1) &= (-\backslash----/-,(\#\#\#!,\Delta),(!\#\#\#,\Delta)) \\ remove(E,1) &= (-\backslash/-,(\#!,\Delta),(!\#,\Delta)) \end{aligned}$$

The operation *insert* will be used to avoid variable clashes (α-conversion), *shift* will be used to preserve the correspondence between a director and its binding, and *remove* will be used to indicate that a β-reduction is done and that a particular director will not be needed further.

In order to make our conversion rules easier to understand, we will use tree representation. Basically, we have two types of rules: the binary rules and the unary rules.

I. Binary Rules The left hand side of each binary rule is of the form:

$$(D,(UdV,E_1,E_2),E_3)$$

where D,U,V are director strings, d is a single director, and the length of D is equal to length of the U, i.e., $|D| = |U|$ (whenever the length of the director string of the left child is bigger than the length of the director string of the parent, we have a redex).

The tree representation is then:

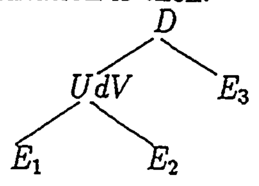


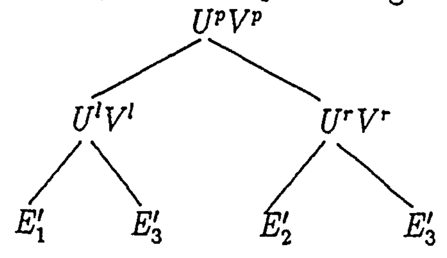
Figure 4

We will use p, r and l for parent, right child, and left child respectively, and give computation rules for evaluating parent, left child, and right child director strings.

I.1

$$(D, (U/\backslash V, E_1, E_2), E_3) \longrightarrow$$
$$(U^pV^p, (U^lV^l, shift(E_1, |U|, |V|), insert(E_3, |U|, |V|)),$$
$$(U^rV^r, shift(E_2, |U|, |V|), insert(E_3, |U|, |V|)))$$

Pictorially, the tree in figure 4 is converted to the tree in figure 5 which says E_1 and E_2 will take E_3 as an argument.

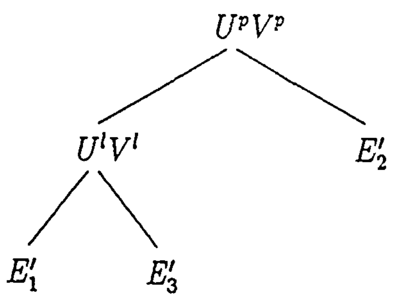


where $E'_1 = shift(E_1, |U|, |V|)$
$E'_2 = shift(E_2, |U|, |V|)$
$E'_3 = insert(E_3, |U|, |V|)$

Figure 5

I.2

$$(D,(U/-V,E_1,E_2),E_3) \longrightarrow$$
$$(U^pV^p,(U^lV^l,shift(E_1,|U|,|V|),insert(E_3,|U|,|V|)),$$
$$remove(E_2,|U|))$$

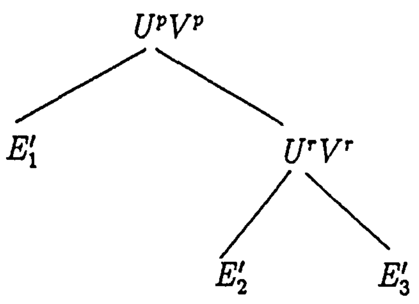


where $E'_1 = shift(E_1,|U|,|V|)$
$E'_2 = remove(E_2,|U|)$
$E'_3 = insert(E_3,|U|,|V|)$

Figure 6

I.3

$$(D,(U-\backslash V,E_1,E_2),E_3) \longrightarrow$$
$$(U^pV^p,remove(E_1,|U|),$$
$$(U^rV^r,shift(E_2,|U|,|V|),insert(E_3,|U|,|V|)))$$



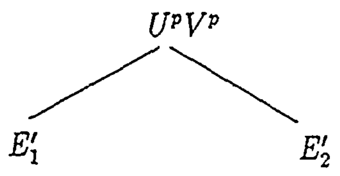
where $E'_1 = remove(E_1,|U|)$
$E'_2 = shift(E_2,|U|,|V|)$
$E'_3 = insert(E_3,|U|,|V|)$

Figure 7

I.4

$$(D,(U--V,E_1,E_2),E_3) \longrightarrow$$

$$(U^pV^p, remove(E_1,|U|), remove(E_2,|U|))$$



where $E_1' = remove(E_1,|U|)$
$E_2' = remove(E_2,|U|)$

Figure 8

II. Unary Rules: The left hand side of each unary rule is of the form

$$(D,(UdV,E_1),E_3) \longrightarrow E_3$$

where D is a binary director string, UdV is an unary director string, and $|D| = |U|$.

II.1

$$(D,(U!V,E_1),E_3) \longrightarrow E_3$$

Pictorially, figure 8 is converted to a tree with one node labelled by E_3.

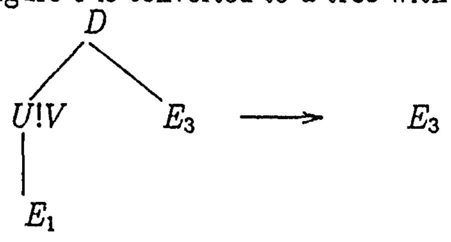


Figure 9

II.2

$$(D,(U\#V,E_1),E_3) \longrightarrow (UV,E_1)$$

Pictorially, in figure 10, the figure on the left is converted to the figure on the right.

$$\begin{tikzcd} & D \arrow[dl, dash] \arrow[dr, dash] & & & UV \arrow[d, dash] \\ U\#V \arrow[d, dash] & & E_3 \arrow[r] & & E_1 \\ E_1 & & & & \end{tikzcd}$$

Figure 10

We now explain in detail how to construct director strings U^pV^p, U^lV^l and U^rV^r. Let $D = d_1d_2\ldots d_k$, $U = u_1u_2\ldots u_k$, and $V = v_1v_2\ldots v_m$. Intuitively d_i's and u_i's refer to the distribution of the variables over left and right expressions in an application. Again referring to figure 4, observe that when $d = /\backslash$, a particular variable occurs in both E_1 and E_2, and hence E_3 will be substituted for that variable in both E_1 and E_2. Also, we point out that after substitution, the variables of E_3 now occur in both (E_1E_3) and (E_2E_3).

Let $d_1, d_2, \ldots, d_k$ refer to variables $x_1, x_2, \ldots, x_k$, then essentially d_i says how x_i is distributed over E_3 and (E_1E_2). If $d_i = q-$ (where q may be $-$ or $/$), then d_i says that x_i does not occur in E_3. Hence after reduction, x_i occurs in (E_1E_3) or (E_2E_3) only if it occurred in E_1 or E_2 before reduction. This gives two cases, depending on whether x_i occurs in E_3 or not:

1. if $d_i = q-$, where $q \in \{-, /\}$, then $u_i^p = u_i$ and U^r and U^l are defined as follows depending on u_i (observe that there may be no U^l or U^r in some cases such as I.2, I.3, or I.4):

(a) $u_i = /\backslash$, then $u_i^l = u_i^r = /-$

(b) $u_i = /-$, then $u_i^l = /-, u_i^r = --$

(c) $u_i = -\backslash$, then $u_i^l = --, u_i^r = /-$

(d) $u_i = --$, then $u_i^l = --, u_i^r = --$

$V^p = V$

2. if $d_i = q\backslash$, where $q \in \{-, /\}$, then U^p is defined as follows depending on d:

   (a) if $d = /\backslash$, then $u_i^p = /\backslash$

   (b) if $d = /-$, then $u_i^p = /-$ when $u_i = s-$ and $u_i^p = /\backslash$ when $u_i = s\backslash$, where $s \in \{-, /\}$.

   (c) if $d = -\backslash$ then $u_i^p = -\backslash$ when $u_i = -s$ and $u_i^p = /\backslash$ when $u_i = \backslash s$, where $s \in \{-, \backslash\}$.

   (d) if $d = --$, then $u_i^p = u_i$

   and U^r and U^l are defined as follows depending on u_i:

   (a) $u_i = /\backslash$, then $u_i^l = u_i^r = /\backslash$

   (b) $u_i = /-$, then $u_i^l = /\backslash, u_i^r = -\backslash$

   (c) $u_i = -\backslash$, then $u_i^l = -\backslash, u_i^r = /\backslash$

   (d) $u_i = --$, then $u_i^l = u_i^r = -\backslash$

$V^p = V$

Now again, let $v_1, v_2, \ldots, v_m$ refer to variables $y_1, y_2 \ldots y_m$. We point out that y_i's represent the list of variables which do not occur in E_3. Here is a complete set of rules for evaluating v_i^l's and v_i^r's:

(a) if $v_i = /\backslash$, then $v_i^l = v_i^r = /-$

(b) if $v_i = /-$, then $v_i^l = /-, v_i^r = --$

(c) if $v_i = -\backslash$, then $v_i^l = --, v_i^r = /-$

(d) if $v_i = --$, then $v_i^l = v_i^r = --$

Example 4.2 *Consider the λ-expression*

$$\lambda f.(\lambda x.\lambda y.x(yf))\lambda p.p$$

or equivalently

$$(/-,(-\backslash/--\backslash,(\#!\#,\Delta),(-\backslash--/-,(\#\#!,\Delta),(!\#\#,\Delta)),(\#!,\Delta))$$

in MDSC. Pictorially, we will show the evaluation in detail. For clarity, we will use asterisk to mark the node where the next reduction occurs.

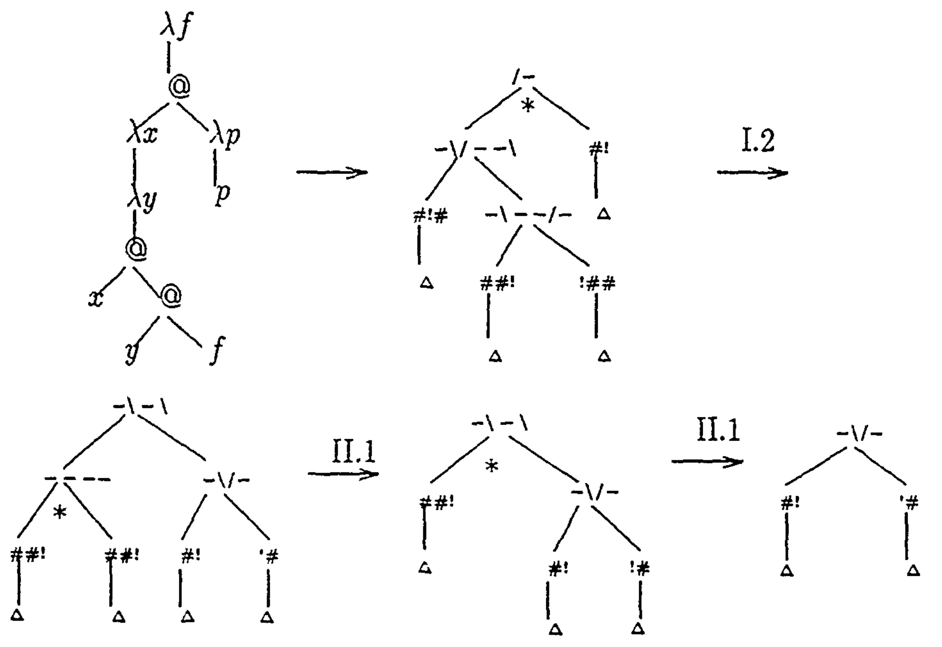


Figure 11

This last tree represents the λ-term $\lambda f \lambda y.(yf)$ which is the normal form of the original λ-expression.

Appendix A gives a reduction for a more detailed example.

Theorem 4.1 *MDSC is confluent.*

Proof: Similar to the proof of theorem 2.3.1 in [8].

□

Remark: Conversion rule I.4 upon application can cause a minor problem which we address here. Consider the λ-expression and its MDS equivalent expression in figure 12.

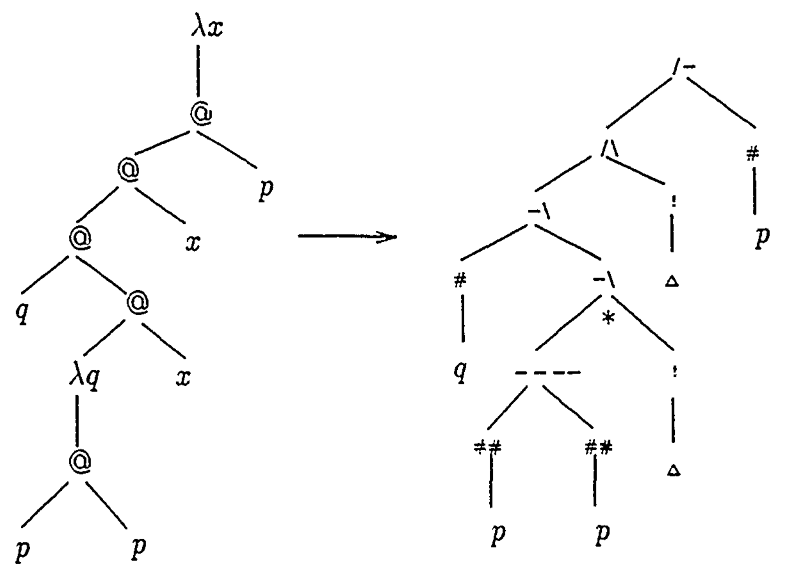


Figure 12

After applying I.4 at $*$, we get the tree in figure 13.

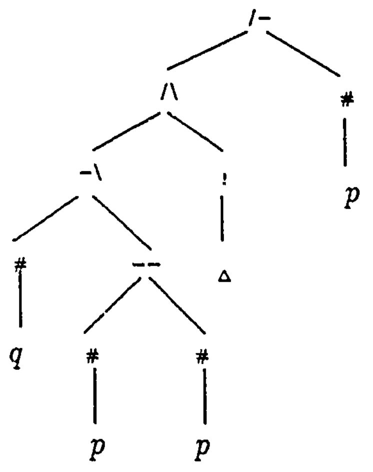


Figure 13

Starting at the root in figure 12, the director $/-$ says that x occurs in the left subtree, the next node labelled by $/\backslash$ says that x occurs both in the left and right subtree, the next node labelled by $-\backslash$ says x occurs in the right subtree, and the next node says x occurs neither in the right subtree nor left subtree. Although it causes no problem when the tree is applied to some argument, it is not representing the same expression in λ-calculus, according to our definition of abstraction rules, after the corresponding β-reduction. The problem is due to the reduction $(\lambda q.pp)x$. The solution is easy, we start at $*$-node in figure 12 and move up the tree towards the root converting each directors of the form $/-$ or $-\backslash$ to $--$. If we encounter a director of the form $/\backslash$, then convert $/\backslash$ to $-\backslash$ or $/-$ depending on the direction we went up the tree and stop. This has to be done for each director d of the type $-\backslash$ at $*$-node. We call this operation *backtracking.* Figure 14 shows the result of this process. Now, we apply rule I.4 at $*$-node in figure 14. Backtracting and rule I.4 has to be done one after another and cannot be separated.

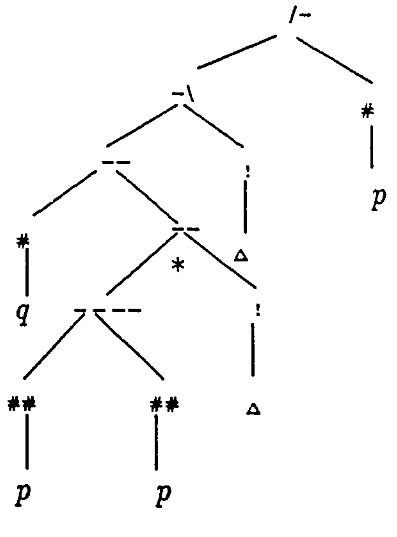


Figure 14

Now we can apply I.4. The same problem can also be caused by # in rule

II.2 and the solution is the same. From now on, we assume the backtracking operation is done when necessary.

5 Translating MDS terms to λ-terms

In this section, we will show how to translate MDS terms back to λ-terms using the following rules:

1. $\Delta \rightarrow \lambda x.x$, where x is a new variable

2. $(!D, \lambda x.x) \rightarrow \lambda x.(D, x)$

3. $(\#D, y) \rightarrow \lambda x.(D, y)$, where x is a new variable

4. $(/\backslash D, \lambda x.E, \lambda y.F) \rightarrow \lambda z.(D, E[x := z], F[y := z])$, where z is a new variable

5. $(/-D, \lambda x.E, \lambda y.F) \rightarrow \lambda z.(D, E[x := z], F)$, where z is a new variable

6. $(-\backslash D, \lambda x.E, \lambda y.F) \rightarrow \lambda z.(D, E, F[y := z])$, where z is a new variable

7. $(--D, \lambda x.E, \lambda y.F) \rightarrow \lambda z.(D, E, F)$ where z is a new variable

Rules (1) to (7) convert MDS terms into MDSλ terms. The following two rules convert MDSλ terms into λ-terms:

1. $(@_1, E) \rightarrow E$

2. $(@_2, E, F) \rightarrow (EF)$

Example 5.1 *Consider the λ-expression*

$$\lambda f.((\lambda x \lambda y.((xf)y))\lambda p.p).$$

The equivalent MDS expression is

$$(/-,(/-/--\backslash,(-\backslash/---,(\#!\#,\Delta),(!\#\#,\Delta)),(\#\#!,\Delta)),(\#!,\Delta))$$

We first translate subterms.

$$\begin{aligned}
(\#!\#,\Delta) &\rightarrow (\#!\#,\lambda x_1.x_1)\\
&\rightarrow \lambda x_2.(!\#,\lambda x_1.x_1)\\
&\rightarrow \lambda x_2 \lambda x_1.(\#,x_1)\\
&\rightarrow \lambda x_2 \lambda x_1 \lambda x_3.(@_1,x_1)
\end{aligned}$$

Similarly,

$$(!\#\#,\Delta) \rightarrow \lambda x_4 \lambda x_5 \lambda x_6(@_1,x_4),$$

and

$$(\#\#!,\Delta) \rightarrow \lambda x_7 \lambda x_8 \lambda x_9(@_1,x_9),$$

and

$$(!\#,\Delta) \rightarrow \lambda x_{10} \lambda x_{11}(@_1,x_{11}).$$

For the subterm

$$\begin{aligned}
& (-\backslash/ - - -, (\#!\#, \Delta), (!\#\#, \Delta)) \\
\rightarrow\; & (-\backslash/ - - -, \lambda x_2 \lambda x_1 \lambda x_3.(@_1, x_1), \lambda x_4 \lambda x_5 \lambda x_6.(@_1, x_4) \\
\rightarrow\; & \lambda x_{12}.(/ - - -, \lambda x_1 \lambda x_3.(@_1, x_1), \lambda x_5 \lambda x_6.(@_1, x_{12}) \\
\rightarrow\; & \lambda x_{12} \lambda x_{13}.(- -, \lambda x_3.(@_1, x_{13}), \lambda x_6.(@_1, x_{12}) \\
\rightarrow\; & \lambda x_{12} \lambda x_{13} \lambda x_{14}.(@_2, (@_1, x_{13}), (@_1, x_{12}))
\end{aligned}$$

Similarly, for the subterm

$$\begin{aligned}
& (/ - / - - \backslash, (-\backslash/ - - -, (\#!\#, \Delta), (!\#\#, \Delta)), (\#\#!, \Delta)) \\
\rightarrow\; & (/ - / - - \backslash, \lambda x_{12} \lambda x_{13} \lambda x_{14}.(@_2, (@_1, x_{13}), (@_1, x_{12})), \lambda x_7 \lambda x_8 \lambda x_9 (@_1, x_9)) \\
\rightarrow\; & \lambda x_{15}.(/ - - \backslash, \lambda x_{13} \lambda x_{14}.(@_2, (@_1, x_{13}), (@_1, x_{15})), \lambda x_8 \lambda x_9 (@_1, x_9)) \\
\rightarrow\; & \lambda x_{15} \lambda x_{16}.(- \backslash, \lambda x_{14}.(@_2, (@_1, x_{16}), (@_1, x_{15})), \lambda x_9 (@_1, x_9)) \\
\rightarrow\; & \lambda x_{15} \lambda x_{16} \lambda x_{17}.(@_2, (@_2, (@_1, x_{16}), (@_1, x_{15})), (@_1, x_{17}))
\end{aligned}$$

And finally,

$$\begin{aligned}
& (/-, (/ - / - - \backslash, (-\backslash/ - - -, (\#!\#, \Delta), (!\#\#, \Delta)), (\#\#!, \Delta)), (\#!, \Delta)) \\
\rightarrow\; & (/-, \lambda x_{15} \lambda x_{16} \lambda x_{17}.(@_2, (@_2, (@_1, x_{16}), (@_1, x_{15})), (@_1, x_{17})), \lambda x_{10} \lambda x_{11} (@_1, x_{11})) \\
\rightarrow\; & \lambda x_{18}.(\lambda x_{16} \lambda x_{17}.(@_2, (@_2, (@_1, x_{16}), (@_1, x_{18})), (@_1, x_{17})), \lambda x_{11} (@_1, x_{11}))
\end{aligned}$$

Translating to λ-calculus,

$$\lambda x_{18}.(\lambda x_{16} \lambda x_{17}.((x_{16} x_{18}) x_{17}) \lambda x_{11}.x_{11}).$$

This is equivalent to the original lambda term up to α-conversion.

Theorem 5.1 *Every MDS expression E has a unique normal form with respect to these rules. Furthermore, for each λ-term E, if we translate E into MDS term, denoted by* $\mathrm{F}(E)$, *and translate* $\mathrm{F}(E)$ *back to λ-term, denoted by* $B(F(E))$, *then* $\mathrm{B}(\mathrm{F}(E)) = E$ *(up to α-conversion).*

Proof:

1. $E = \lambda x_1 \ldots x_n.x$

$$\begin{aligned}
\mathrm{B}(\mathrm{F}(E)) &= \mathrm{B}((D,\Delta)) \ \textit{where}\ D = \#^{i-1}!\#^{n-i}\ \textit{if}\ x = x_i \\
\mathrm{B}(\mathrm{F}(E)) &= \mathrm{B}((D,x)) \ \textit{where}\ D = \#^{n}\ \textit{if no}\ x = x_i \\
&= E\ \ (\textit{upto}\ \alpha - \textit{conversion})
\end{aligned}$$

2. $E = \lambda x_1 \ldots x_n.(FG)$

First consider $\mathrm{F}(E)$. First, we rewrite $\lambda x_n.(FG)$ giving $(d_n, \lambda x_n.F, \lambda x_n.G)$. d_n is $/\backslash, /-, -\backslash$, or $--$, depending upon the free occurrence of x in F and/or G. Next, we reduce

$$\lambda x_{n-1}(d_n, \lambda x_n F, \lambda x_n G)$$

giving

$$(d_{n-1}d_n, \lambda x_{n-1}x_n.F, \lambda x_{n-1}x_n.G).$$

Continuing this process for all lambdas in the outer most parentheses, we get

$$\begin{aligned}
\mathrm{F}(E) &= \mathrm{F}((D, \lambda x_1 \ldots x_n.F, \lambda x_1 \ldots x_n.G)) \\
&= (D, \mathrm{F}(\lambda x_1 \ldots x_n.F), \mathrm{F}(\lambda x_1 \ldots x_n.G)).
\end{aligned}$$

Hence, $B(F(E)) = B((D, F(\lambda x_1 \ldots x_n.F), F(\lambda x_1 \ldots x_n.G))$. By confluence of the B-rules, we can evaluate the right hand side by first evaluating each subexpression giving

$$\begin{aligned} B(F(E)) &= B((D, F(\lambda x_1 \ldots x_n.F), F(\lambda x_1 \ldots x_n.G)) \\ &= B((D, B(F(\lambda x_1 \ldots x_n.F)), B(F(\lambda x_1 \ldots x_n.G))) \\ &= B((D, \lambda x_1 \ldots x_n.F, \lambda x_1 \ldots x_n.G)). \end{aligned}$$

We can now apply B-rules to eliminate D, giving

$$B((D, \lambda x_1 \ldots x_n.F, \lambda x_1 \ldots x_n.G)) = \lambda x_1 \ldots x_n.(FG) = E.$$

□

6 Conclusion

We have modified director string calculus to obtain strong reduction as opposed to weak reduction given in [8]. in addition, this modified calculus can be considered as a different implementation of λ-calculus. Particularly, the subsitution operation supports the ideas given by Revesz in [13] and O'Donnell and Strandh in [11].

MDSC provides an environment in which, one can further study and implement optimal reduction and parallel reduction [4, 7, 9, 10, 14]. There is also a close relationship between the position of a director in the director string and de Bruijn number [2]. This is basically the way both MDSC and de

Bruijn calculus avoid α-conversion. We intend to point out these connections in detail in our upcoming paper.

References

[1] BARENDREGT, H. P. *The Lambda Calculus - its syntax and semantics.* North Holland, 1981, 1984.

[2] DE BRUIJN, N. G. Lambda calculus notation with nameless dummies. *Inagationes Mathematicae 34* (1972), 381–392.

[3] DIJKSTRA, E. W. A mild variant of combinatory logic. Unpublished note. EWD735, 1980.

[4] FIELD, J., AND TEITELBAUM, T. On laziness and optimality in lambda interpreters: Tools for specification and analysis. In *Proceedings of Principles of Programming Languages* (January 1990), ACM, ACM Press.

[5] HINDLEY, J. R., LERCHER, B., AND SELDIN, J. P. *Introduction to combinatory logic.* Cambridge University Press, 1972.

[6] HUDAK, P., AND KRANZ, D. A combinator-based compiler for a functional language. In *11th ACM Symposium on Principles of Programming Languages* (1984), ACM, pp. 121–132.

[7] HUDAK, P., AND MOHR, E. Graphinators and the duality of SIMD and MIMD. In *Proceedings 1988 ACM Conference on Lisp and Functional Programming* (1988), pp. 224–234.

[8] KENNAWAY, R., AND SLEEP, R. Director strings as combinators. *ACM Trans. on Programming Languages and Systems 10*, 4 (October 1988), 602–626.

[9] LAMPING, J. An algorithm for optimal lambda calculus reduction. In *Proceedings of Principles of Programming Languages* (January 1990), ACM. See also his earlier draft.

[10] LEVY, J. J. Optimal reductions in the lambda-calculus. In *To H. B. Curry: Essays on Combinatory logic, Lambda Calculus, and Formalism.* Academic Press, 1980.

[11] O'DONNELL, M. J., AND STRANDH, R. I. Toward a fully parallel implementation of the lambda calculus. Tech. Rep. JHU/EECS-84/13, The Johns Hopkins University, 1984.

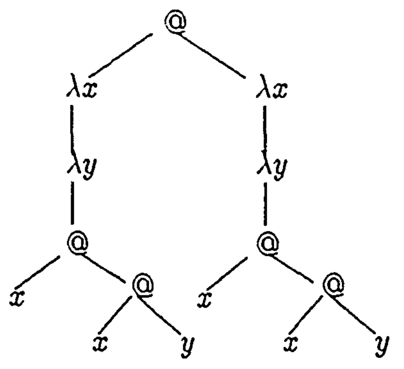
[12] PEYTON-JONES, S. *The Implementation of Functional Programming Languages.* Prentice-Hall International, Englewood Cliffs, NJ, 1987.

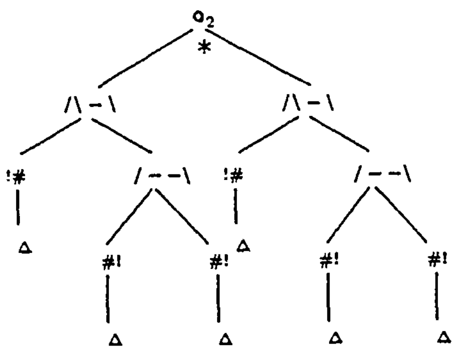
[13] REVESZ, G. Axioms for the theory of lambda-conversion. *SIAM Journal of Computing 14*, 2 (May 1985).

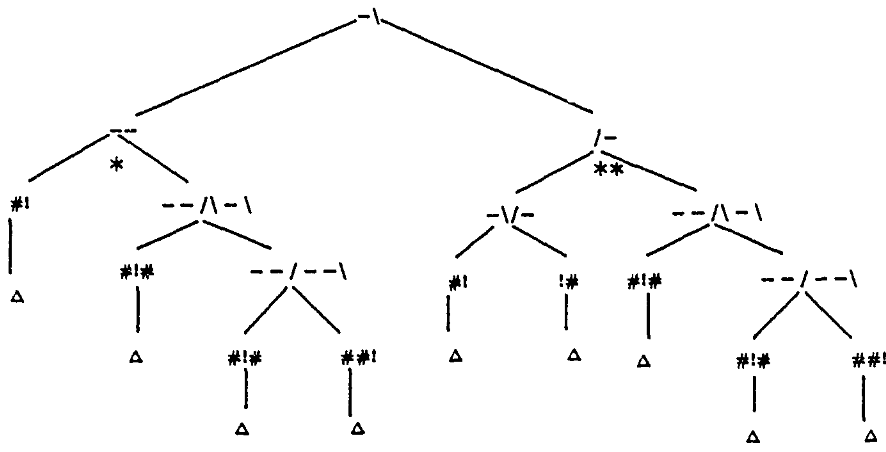
[14] STAPLES, J. Optimal evaluations of graph-like expressions. *Theoretical Computer Science 10* (1980), 297–316.

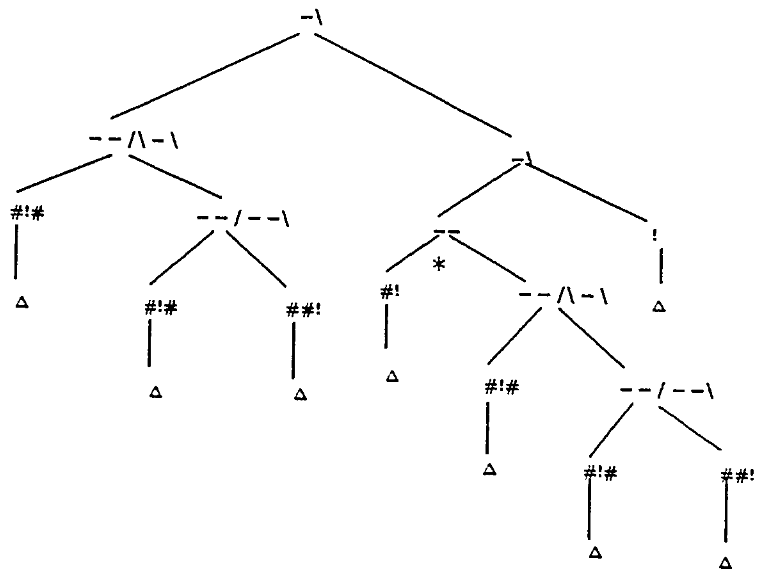
[15] TURNER, D. A. A new implementation technique for applicative languages. *Software-Practice and Experience 9* (1979), 31–49.

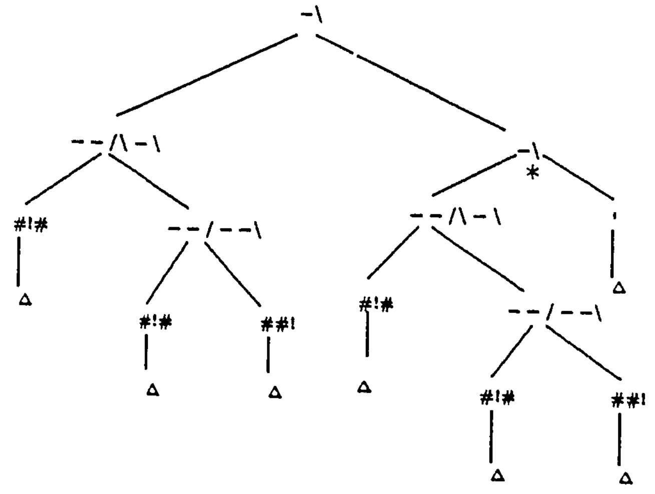
A Church's Numeral

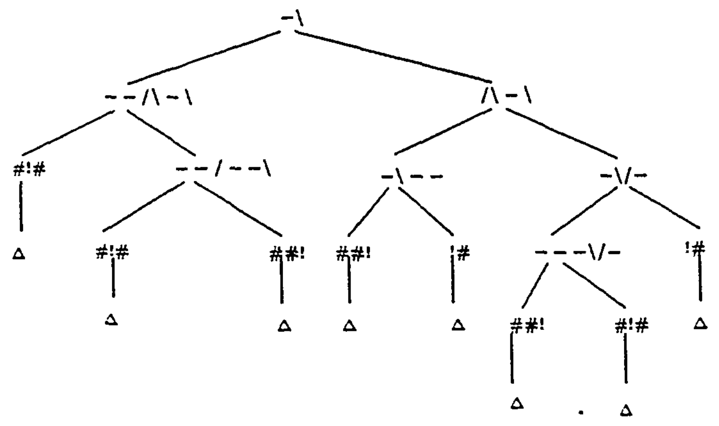




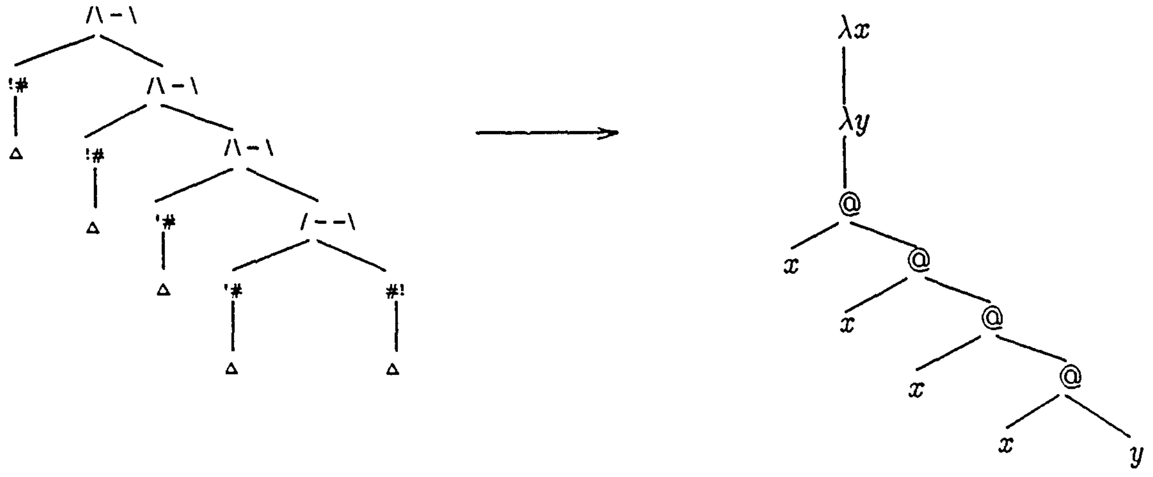








After few more steps we get


/\-\
!#
/\-\
!#
/\-\
!#
/--\
!#
#!
λx
λy
@
x
@
x
@
x
@
x
y